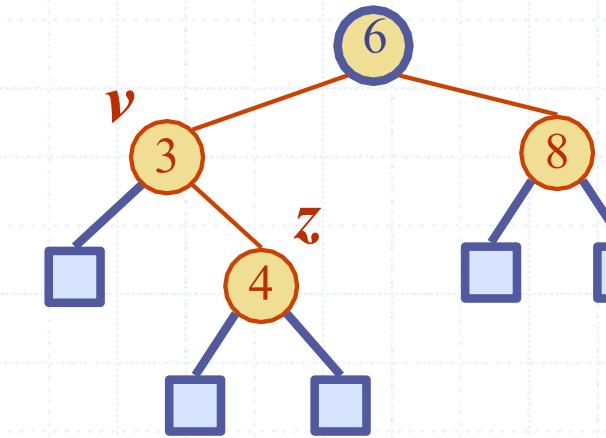


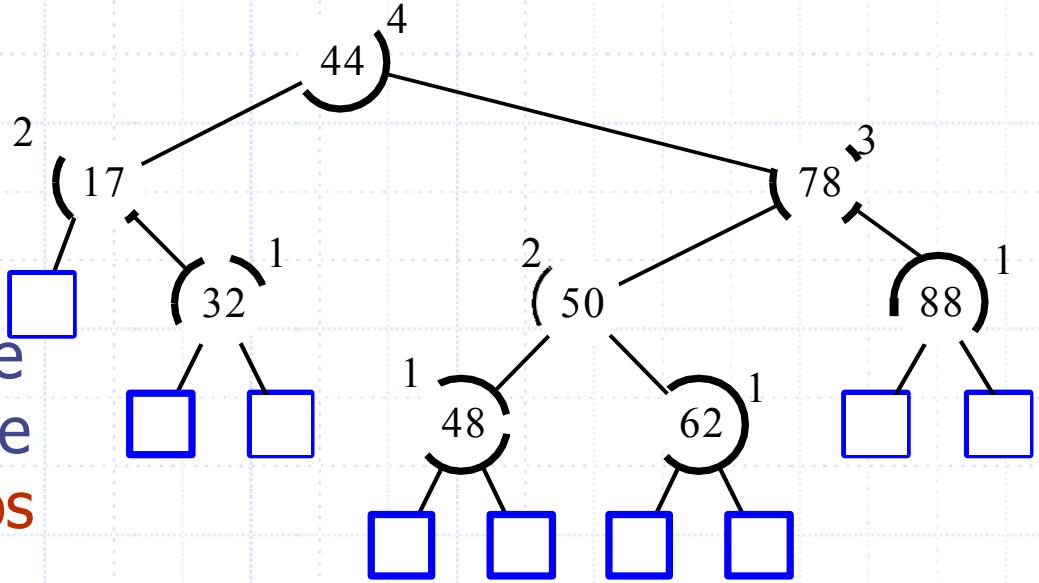
Presentación para usar con el libro de texto **Diseño y aplicaciones de algoritmos**, de MT Goodrich y R. Tamassia, Wiley, 2015

Árboles AVL



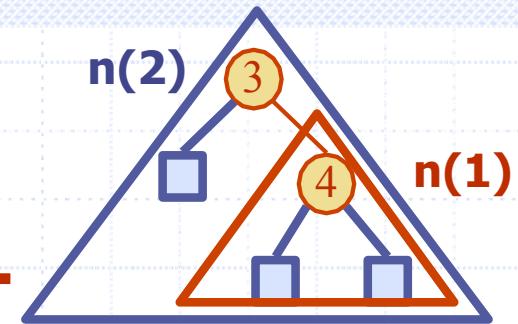
Árbol AVL Definición

- ◆ Los árboles AVL son árboles balanceados por rango
- ◆ El **rango**, $r(v)$, de cada nodo, v , es su altura.
- ◆ **Regla de equilibrio:**
Un árbol AVL es un árbol binario de búsqueda tal que para cada nodo interno v de T , las alturas (rangos) de los hijos de v no pueden diferir como máximo 1.



Un ejemplo de un árbol AVL donde el
Los rangos se muestran al lado
del nodos

Altura de una Árbol AVL



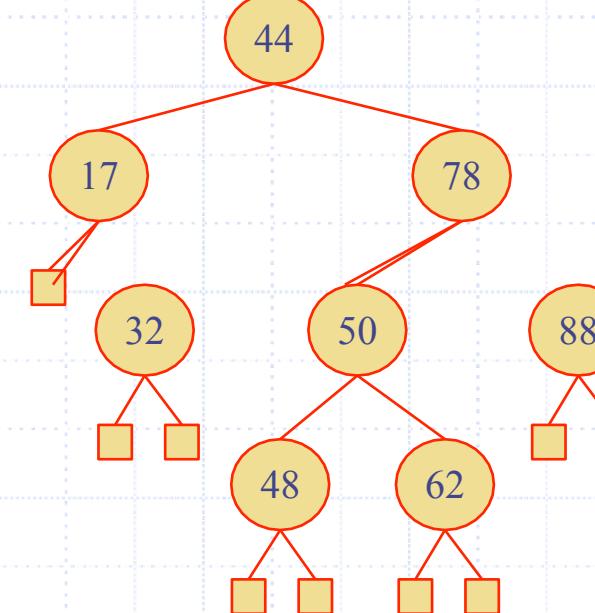
La altura de un árbol AVL que almacena n claves es $O(\log n)$.

Prueba (por inducción) : limitemos $n(h)$: el número mínimo de nodos internos de un árbol AVL de altura h .

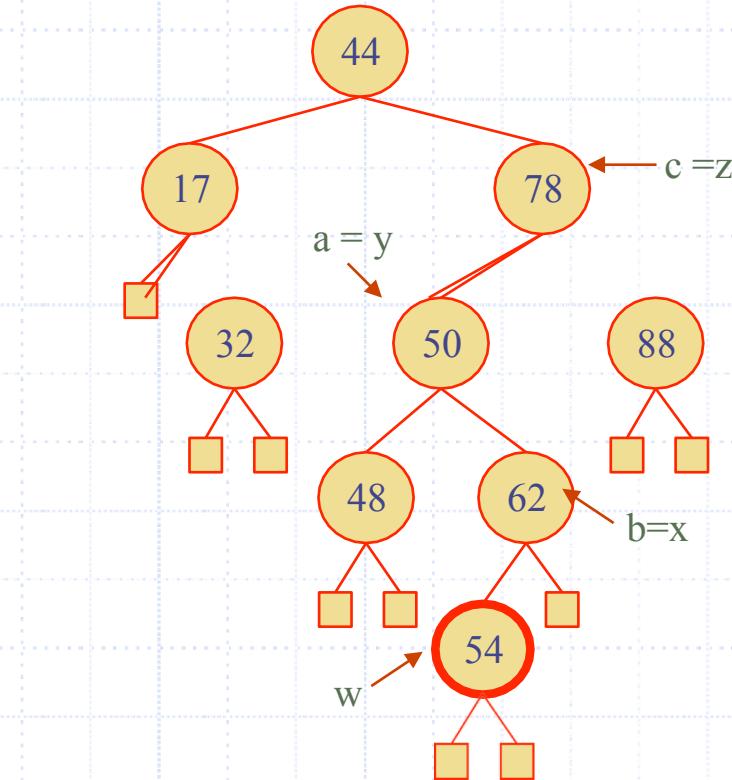
- Vemos fácilmente que $n(1) = 1$ y $n(2) = 2$
- Para $n > 2$, un árbol AVL de altura h contiene el nodo raíz , un subárbol AVL de altura $n-1$ y otro de altura $n-2$.
- Es decir, $n(h) = 1 + n(h-1) + n(h-2)$
- Sabiendo que $n(h-1) > n(h-2)$, obtenemos $n(h) > 2n(h-2)$. Entonces, $n(h) > 2n(h-2)$, $n(h) > 4n(h-4)$, $n(h) > 8n(h-6)$, ... (por inducción), $n(h) > 2^i n(h-2i)$
- Resolviendo el caso base obtenemos : $n(h) > 2^{h/2-1}$
- Tomando logaritmos: $h < 2\log n(h) + 2$
- Entonces, la altura de un árbol AVL es $O(\log n)$

Inserción

- ◆ La inserción es como en un árbol binario de búsqueda.
- ◆ Siempre se hace expandiendo un nodo externo .
- ◆ Ejemplo:



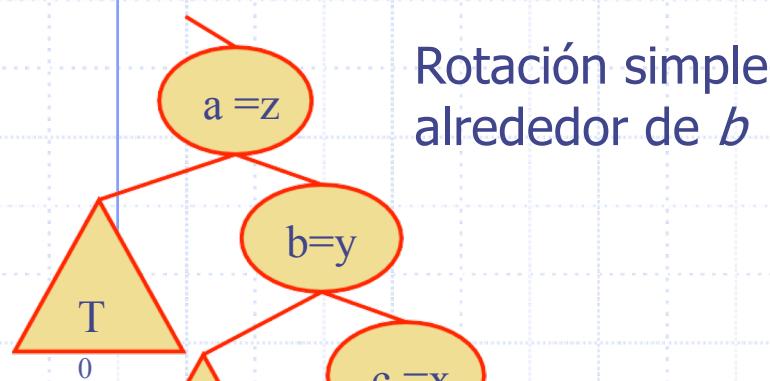
antes de la inserción



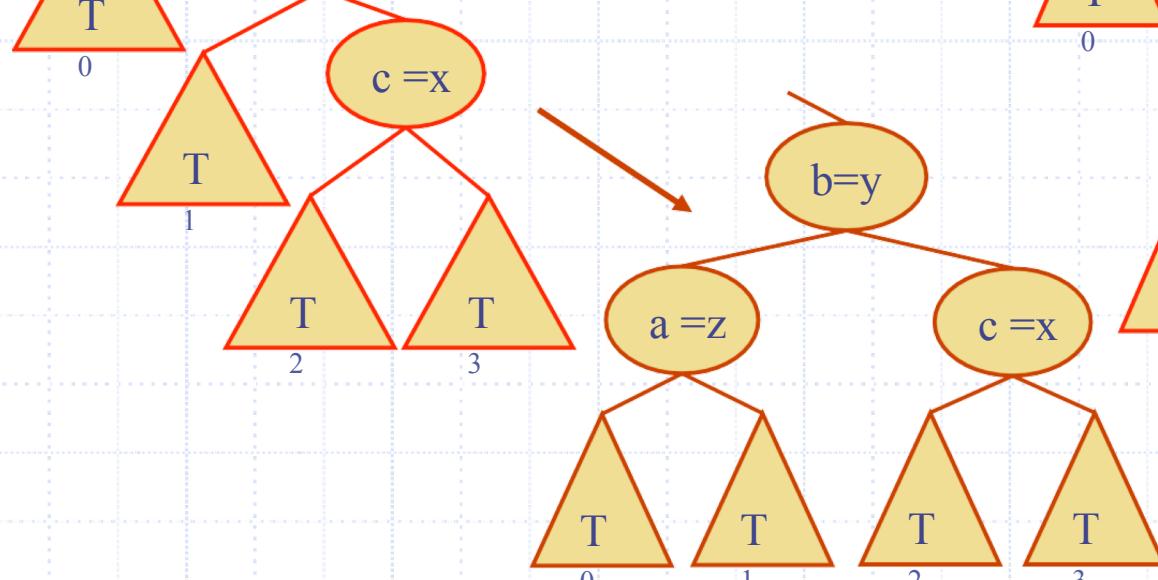
después de la inserción

Reestructuración de 3-Nodos

- Sean (a, b, c) la lista desordenada de x, y, z
- Realice las rotaciones necesarias para hacer que b sea el nodo más superior de los tres

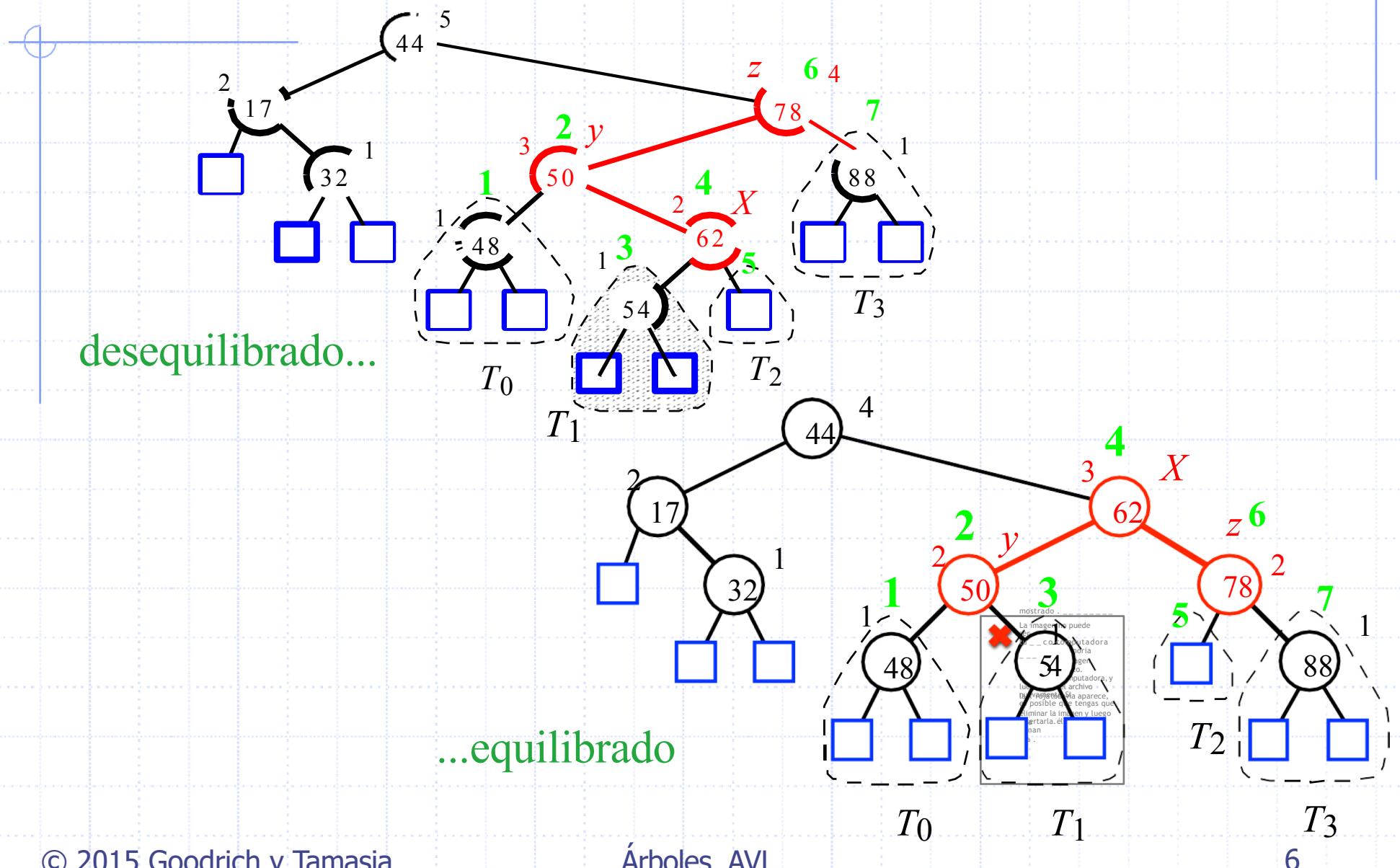


Rotación simple
alrededor de b



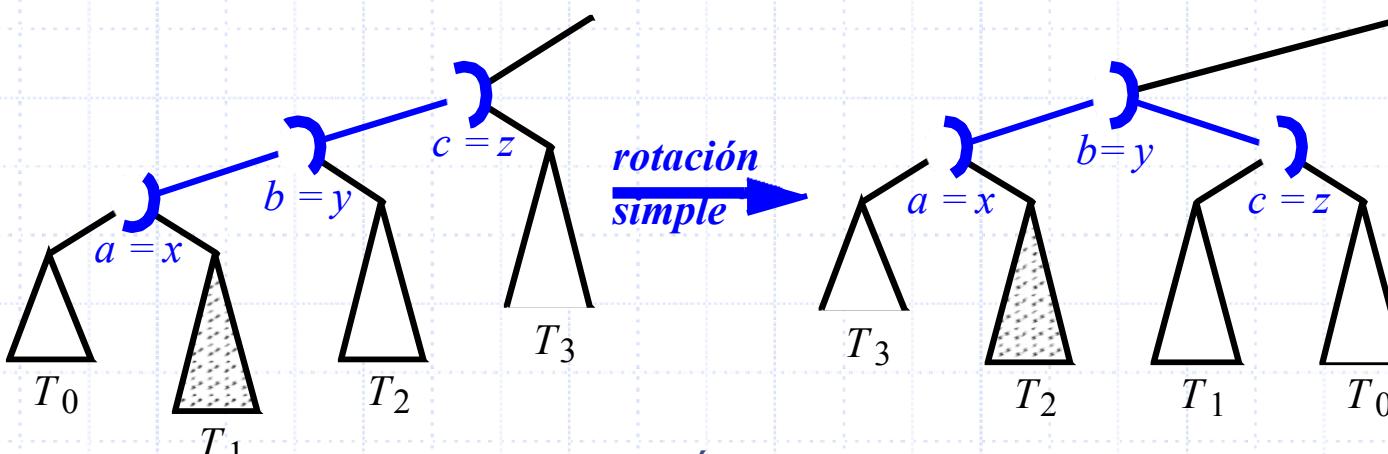
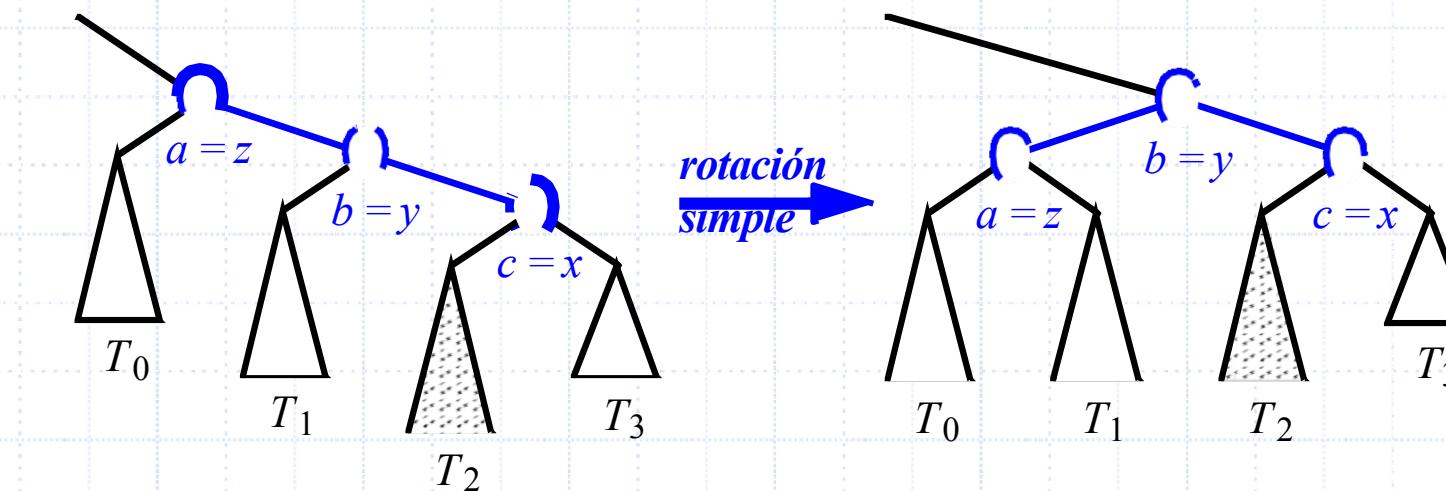
Doble rotación alrededor
alrededor de c y a

Ejemplo de inserción (cont.)



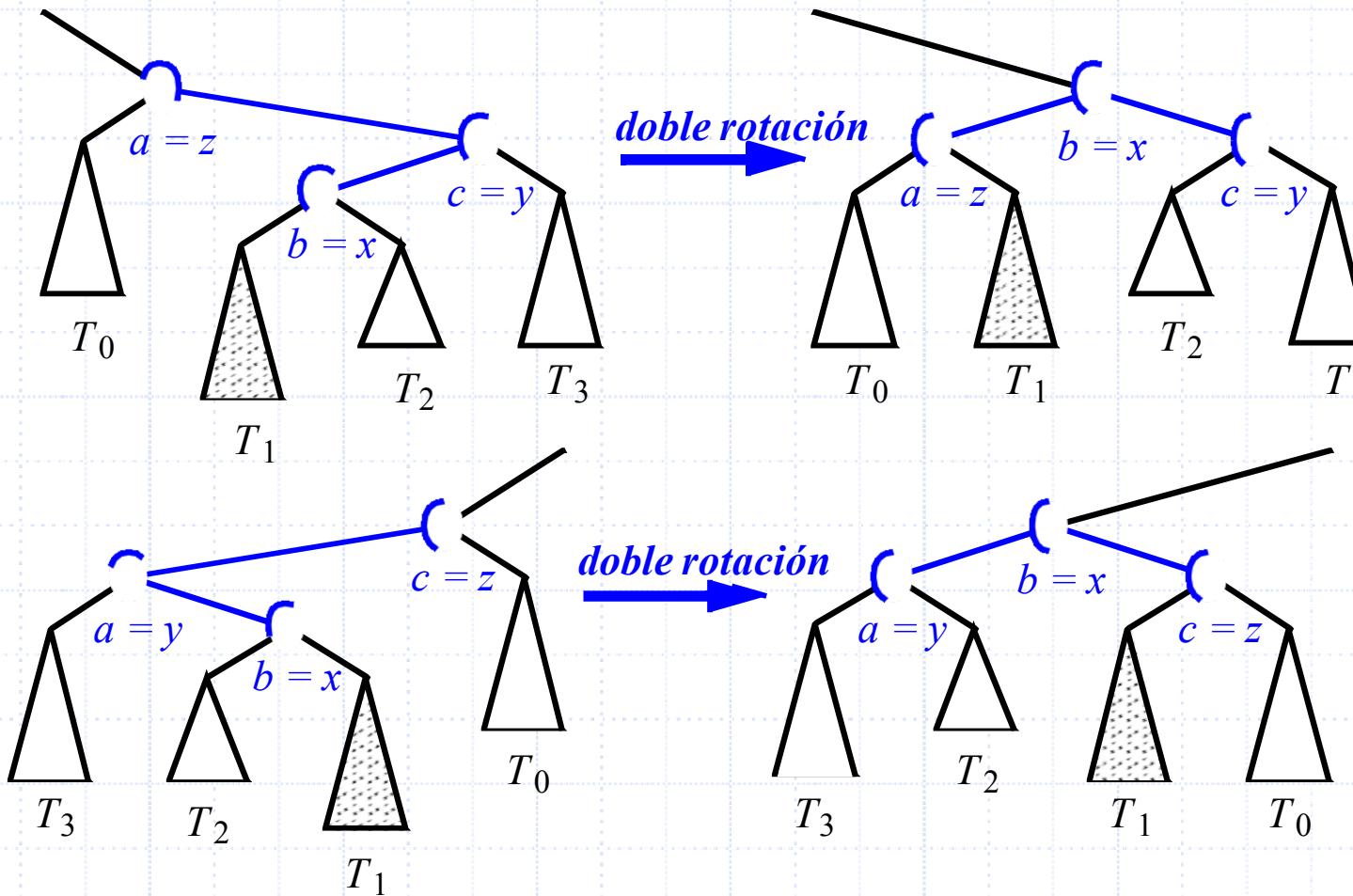
Reestructuración (Rotaciones Simples)

◆ Rotaciones Simples:



Reestructuración (Rotaciones dobles)

◆ Rotaciones dobles:



Pseudocódigo

◆ Inserción

Algorithm insertAVL(k, e, T):

Input: A key-element pair, (k, e) , and an AVL tree, T

Output: An update of T to now contain the item (k, e)

$v \leftarrow \text{IterativeTreeSearch}(k, T)$

if v is not an external node **then**

return “An item with key k is already in T ”

Expand v into an internal node with two external-node children

$v.\text{key} \leftarrow k$

$v.\text{element} \leftarrow e$

$v.\text{height} \leftarrow 1$

$\text{rebalanceAVL}(v, T)$

Pseudocódigo

◆ Reequilibrar en un nodo que viola el equilibrio.

Algorithm rebalanceAVL(v, T):

Input: A node, v , where an imbalance may have occurred in an AVL tree, T

Output: An update of T to now be balanced

$v.height \leftarrow 1 + \max\{v.leftChild().height, v.rightChild().height\}$

while v is not the root of T **do**

$v \leftarrow v.parent()$

if $|v.leftChild().height - v.rightChild().height| > 1$ **then**

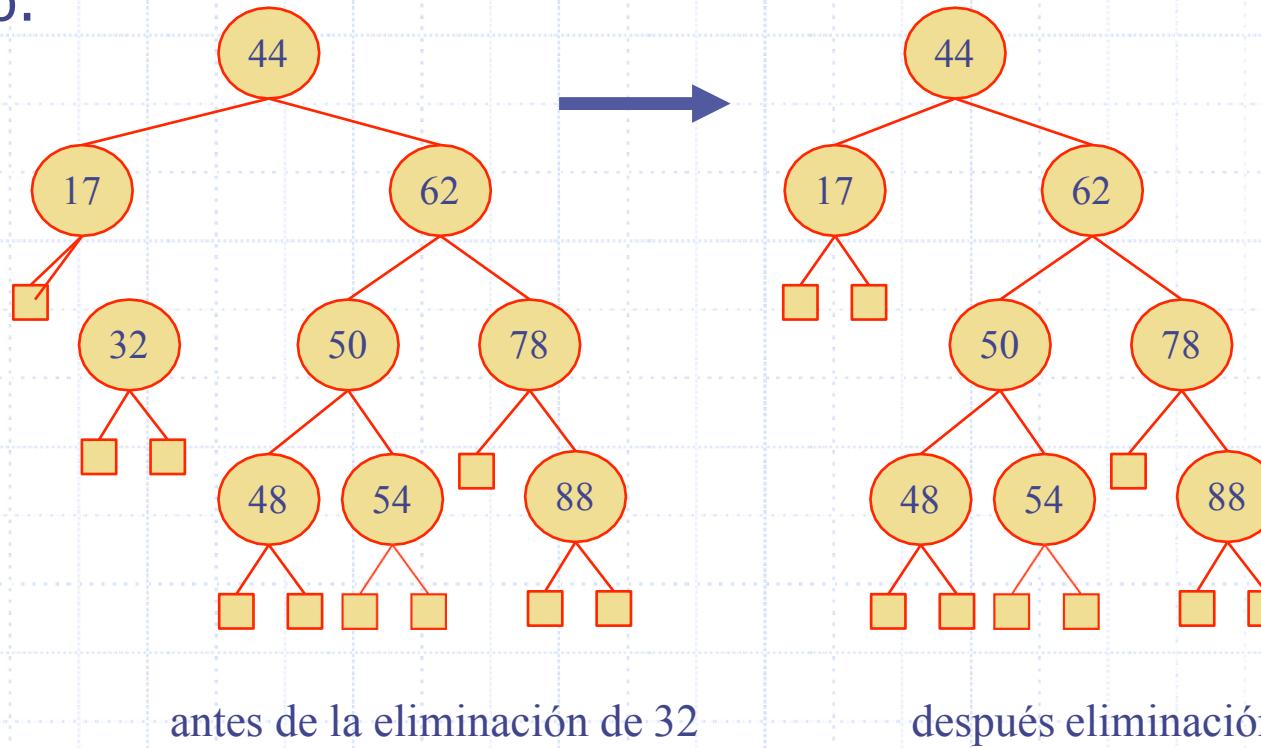
 Let y be the tallest child of v and let x be the tallest child of y

$v \leftarrow \text{restructure}(x)$ // trinode restructure operation

$v.height \leftarrow 1 + \max\{v.leftChild().height, v.rightChild().height\}$

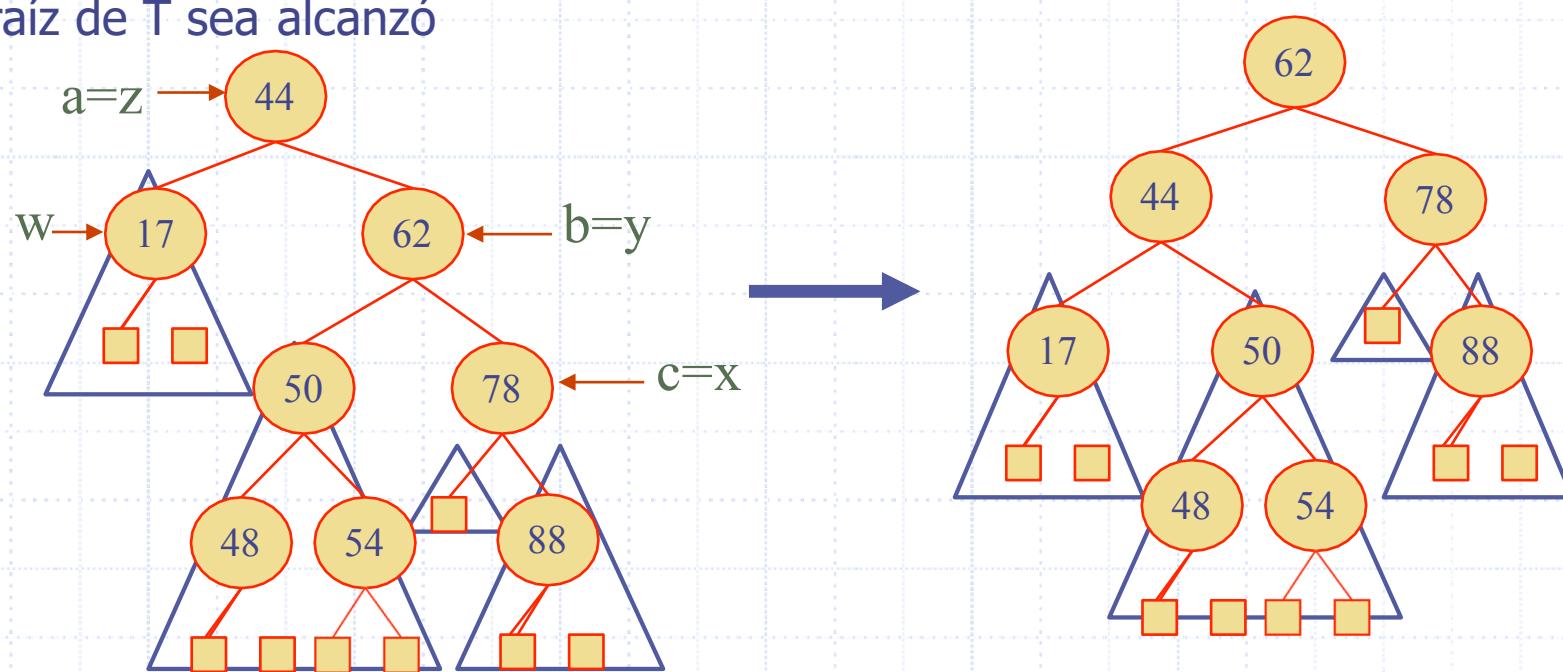
Eliminación

- ◆ La eliminación comienza como en un árbol de búsqueda binario , lo que significa que el nodo eliminado se convertirá en un nodo externo vacío. Su padre, w, puede causar una desequilibrio.
 - ◆ Ejemplo:



Rebalanceo después de una Eliminación

- ◆ Sea z el primer nodo desequilibrado encontrado mientras sube por el árbol desde w . Además, sea y el hijo de z con mayor altura, y sea x el hijo de y con mayor altura . altura
- ◆ Realizamos una **reestructuración de 3-nodos** para restablecer el equilibrio en z
- ◆ Como esta reestructuración puede alterar el equilibrio de otro nodo situado más arriba en el árbol, debemos seguir comprobando el equilibrio hasta que la raíz de T sea alcanzó



Pseudocódigo

◆ Eliminación

Algorithm removeAVL(k, T):

Input: A key, k , and an AVL tree, T

Output: An update of T to now have an item (k, e) removed

$v \leftarrow \text{IterativeTreeSearch}(k, T)$

if v is an external node **then**

return “There is no item with key k in T ”

if v has no external-node child **then**

Let u be the node in T with key nearest to k

Move u ’s key-value pair to v

$v \leftarrow u$

Let w be v ’s smallest-height child

Remove w and v from T , replacing v with w ’s sibling, z

rebalanceAVL(z, T)

Performance Árbol AVL

- ◆ Árbol AVL que almacena n elementos
 - La estructura de datos utiliza espacio $O(n)$
 - Una sola reestructuración requiere tiempo $O(1)$
 - ◆ usando un árbol binario enlazado
 - La búsqueda toma tiempo $O(\log n)$
 - ◆ la altura del árbol es $O(\log n)$, sin reestructuraciones
 - La inserción toma tiempo $O(\log n)$
 - ◆ hallazgo inicial es $O(\log n)$
 - ◆ reestructurar el árbol, mantener las alturas es $O(\log n)$
 - La eliminación toma tiempo $O(\log n)$
 - ◆ hallazgo inicial es $O(\log n)$
 - ◆ reestructurar el árbol, mantener las alturas es $O(\log n)$